LSTM System Identification

Name: Ananda Cahyo Wibowo

NRP : 07111940000128

Undergrad Thesis Title : Data Driven Gas Lift Well And Network Optimization With Neural Network Based System Identification Using Modbus Simulator

Data Preparation

In [ ]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
#from datetime import datetime

#Read the csv file
df = pd.read_csv("upsample_min.csv")
df = pd.read_csv("upsample.csv")
df = pd.read_csv("upsampled_matlab.csv")
df2=df.drop(df.columns[0], axis=1)
data = df['glir11'].to_numpy()

split = 0.75

x1 = df['glir11'].to_numpy()[:int(split*len(data))]
x1 = x1.reshape(len(x1),1)
y1 = df['qo11'].to_numpy()[:int(split*len(data))]
y1 = y1.reshape(len(y1),1)

x2 = df['glir11'].to_numpy()[int(split*len(data)):]
y2 = df['qo11'].to_numpy()[int(split*len(data)):]
```

```
print(f"ukuran x train: {np.shape(x1)} ukuran y train: {np.shape(y1)}")
print(f"ukuran x test: {np.shape(x2)} ukuran y test: {np.shape(y2)}")
```

```
c:\Users\ASUS\AppData\Local\Programs\Python\Python39\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy vers
ion >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
ukuran x train: (11400, 1) ukuran y train: (11400, 1)
ukuran x test: (3801,) ukuran y test: (3801,)
```

Train Data

Preprocessing Data

In [ ]:

```
#New dataframe with only training data
df_for_training_x = x1
df_for_training_y = y1

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_training_x)
scaler2 = scaler.fit(df_for_training_y)
df_for_training_scaled_x = scaler.transform(df_for_training_x)
df_for_training_scaled_y = scaler2.transform(df_for_training_y)

#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 5. We will make timesteps = 14 (past days data used for training).

#Empty lists to be populated using formatted training data
trainX = []
trainY = []

n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_training_scaled has a shape (12823, 5)
#12823 refers to the number of data points and 5 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_training_scaled_x) - n_future +1):
    trainX.append(df_for_training_scaled_x[i - n_past:i, 0:df_for_training_x.shape[1]])
```

```python
for i in range(n_past, len(df_for_training_scaled_y) - n_future +1):
    trainY.append(df_for_training_scaled_y[i + n_future - 1:i + n_future, 0])

trainX, trainY = np.array(trainX), np.array(trainY)

print('trainX shape == {}.'.format(trainX.shape))
print('trainY shape == {}.'.format(trainY.shape))
```

```
trainX shape == (11386, 14, 1).
trainY shape == (11386, 1).
```

RNN LSTM Architecture & Training

In [ ]:

```python
# define the Autoencoder model

model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse')
model.summary("")


# fit the model
history = model.fit(trainX, trainY, epochs=100, batch_size=15, validation_split=0.1, verbose=1)

model.save("RNN_model_resolved")
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 14, 64)            16896

 lstm_1 (LSTM)               (None, 14, 64)            33024

 lstm_2 (LSTM)               (None, 32)                12416

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 62,369
Trainable params: 62,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
684/684 [==============================] - 43s 45ms/step - loss: 0.4824 - val_loss: 0.1940
Epoch 2/100
684/684 [==============================] - 25s 37ms/step - loss: 0.4120 - val_loss: 0.1741
Epoch 3/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3946 - val_loss: 0.1630
Epoch 4/100
684/684 [==============================] - 31s 46ms/step - loss: 0.4123 - val_loss: 0.1773
Epoch 5/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3857 - val_loss: 0.1496
Epoch 6/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3762 - val_loss: 0.1689
Epoch 7/100
684/684 [==============================] - 30s 44ms/step - loss: 0.3743 - val_loss: 0.2033
Epoch 8/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3757 - val_loss: 0.1258
Epoch 9/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3706 - val_loss: 0.1151
Epoch 10/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3699 - val_loss: 0.1883
Epoch 11/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3673 - val_loss: 0.1533
Epoch 12/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3726 - val_loss: 0.1688
```

```
Epoch 13/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3687 - val_loss: 0.1390
Epoch 14/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3681 - val_loss: 0.1772
Epoch 15/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3629 - val_loss: 0.1502
Epoch 16/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3585 - val_loss: 0.1504
Epoch 17/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3613 - val_loss: 0.1495
Epoch 18/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3602 - val_loss: 0.1715
Epoch 19/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3569 - val_loss: 0.2129
Epoch 20/100
684/684 [==============================] - 29s 43ms/step - loss: 0.3595 - val_loss: 0.2042
Epoch 21/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3615 - val_loss: 0.1306
Epoch 22/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3534 - val_loss: 0.1461
Epoch 23/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3511 - val_loss: 0.1328
Epoch 24/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3534 - val_loss: 0.1565
Epoch 25/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3599 - val_loss: 0.1808
Epoch 26/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3482 - val_loss: 0.1708
Epoch 27/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3456 - val_loss: 0.1544
Epoch 28/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3523 - val_loss: 0.1749
Epoch 29/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3486 - val_loss: 0.2199
Epoch 30/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3494 - val_loss: 0.1780
Epoch 31/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3524 - val_loss: 0.1507
Epoch 32/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3425 - val_loss: 0.1587
Epoch 33/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3421 - val_loss: 0.2174
Epoch 34/100
```

```
684/684 [==============================] - 31s 46ms/step - loss: 0.3575 - val_loss: 0.2057
Epoch 35/100
684/684 [==============================] - 26s 38ms/step - loss: 0.3772 - val_loss: 0.1677
Epoch 36/100
684/684 [==============================] - 24s 36ms/step - loss: 0.3595 - val_loss: 0.1894
Epoch 37/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3535 - val_loss: 0.1617
Epoch 38/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3463 - val_loss: 0.1848
Epoch 39/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3670 - val_loss: 0.2073
Epoch 40/100
684/684 [==============================] - 31s 46ms/step - loss: 0.3638 - val_loss: 0.1483
Epoch 41/100
684/684 [==============================] - 31s 45ms/step - loss: 0.3520 - val_loss: 0.1517
Epoch 42/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3438 - val_loss: 0.1746
Epoch 43/100
684/684 [==============================] - 11s 17ms/step - loss: 0.3380 - val_loss: 0.1674
Epoch 44/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3383 - val_loss: 0.1653
Epoch 45/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3287 - val_loss: 0.2120
Epoch 46/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3332 - val_loss: 0.1614
Epoch 47/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3214 - val_loss: 0.1434
Epoch 48/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3220 - val_loss: 0.1663
Epoch 49/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3131 - val_loss: 0.1572
Epoch 50/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3288 - val_loss: 0.2725
Epoch 51/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3535 - val_loss: 0.1595
Epoch 52/100
684/684 [==============================] - 13s 18ms/step - loss: 0.4078 - val_loss: 0.1368
Epoch 53/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3336 - val_loss: 0.1457
Epoch 54/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3382 - val_loss: 0.1816
Epoch 55/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3336 - val_loss: 0.1488
```

```
Epoch 56/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3256 - val_loss: 0.1450
Epoch 57/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3210 - val_loss: 0.1297
Epoch 58/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3230 - val_loss: 0.1740
Epoch 59/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3255 - val_loss: 0.1581
Epoch 60/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3149 - val_loss: 0.1460
Epoch 61/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3301 - val_loss: 0.1741
Epoch 62/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3163 - val_loss: 0.1621
Epoch 63/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3181 - val_loss: 0.1387
Epoch 64/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3180 - val_loss: 0.1536
Epoch 65/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3278 - val_loss: 0.1570
Epoch 66/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3297 - val_loss: 0.1493
Epoch 67/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3239 - val_loss: 0.1756
Epoch 68/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3253 - val_loss: 0.1553
Epoch 69/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3172 - val_loss: 0.1627
Epoch 70/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3199 - val_loss: 0.1484
Epoch 71/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3129 - val_loss: 0.1778
Epoch 72/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3292 - val_loss: 0.1530
Epoch 73/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3283 - val_loss: 0.1319
Epoch 74/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3402 - val_loss: 0.1619
Epoch 75/100
684/684 [==============================] - 13s 19ms/step - loss: 0.5604 - val_loss: 0.3825
Epoch 76/100
684/684 [==============================] - 13s 18ms/step - loss: 2.9325 - val_loss: 0.1817
Epoch 77/100
```

```
684/684 [==============================] - 12s 18ms/step - loss: 0.3764 - val_loss: 0.1716
Epoch 78/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3417 - val_loss: 0.1480
Epoch 79/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3917 - val_loss: 0.1462
Epoch 80/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3513 - val_loss: 0.1509
Epoch 81/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3443 - val_loss: 0.1510
Epoch 82/100
684/684 [==============================] - 11s 16ms/step - loss: 0.3854 - val_loss: 0.1679
Epoch 83/100
684/684 [==============================] - 13s 18ms/step - loss: 0.3794 - val_loss: 0.1560
Epoch 84/100
684/684 [==============================] - 12s 18ms/step - loss: 0.4583 - val_loss: 0.1528
Epoch 85/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3702 - val_loss: 0.1542
Epoch 86/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3529 - val_loss: 0.1842
Epoch 87/100
684/684 [==============================] - 12s 17ms/step - loss: 0.3638 - val_loss: 0.1842
Epoch 88/100
684/684 [==============================] - 12s 18ms/step - loss: 0.3349 - val_loss: 0.1519
Epoch 89/100
684/684 [==============================] - 13s 19ms/step - loss: 0.3481 - val_loss: 0.2136
Epoch 90/100
684/684 [==============================] - 12s 17ms/step - loss: 0.4806 - val_loss: 0.2932
Epoch 91/100
684/684 [==============================] - 12s 18ms/step - loss: 0.4048 - val_loss: 0.1840
Epoch 92/100
684/684 [==============================] - 12s 18ms/step - loss: 0.4209 - val_loss: 0.1961
Epoch 93/100
684/684 [==============================] - 12s 17ms/step - loss: 1.4018 - val_loss: 0.2078
Epoch 94/100
684/684 [==============================] - 11s 16ms/step - loss: 0.6062 - val_loss: 0.2405
Epoch 95/100
684/684 [==============================] - 12s 18ms/step - loss: 0.4509 - val_loss: 0.2259
Epoch 96/100
684/684 [==============================] - 12s 18ms/step - loss: 0.7560 - val_loss: 0.2617
Epoch 97/100
684/684 [==============================] - 12s 18ms/step - loss: 0.5470 - val_loss: 0.1656
Epoch 98/100
684/684 [==============================] - 12s 18ms/step - loss: 0.5250 - val_loss: 0.2404
```

```
Epoch 99/100
684/684 [==============================] - 11s 17ms/step - loss: 0.5197 - val_loss: 0.2071
Epoch 100/100
684/684 [==============================] - 12s 18ms/step - loss: 0.5691 - val_loss: 0.2035
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: RNN_model_resolved\assets
INFO:tensorflow:Assets written to: RNN_model_resolved\assets
```

Weights and Biasses

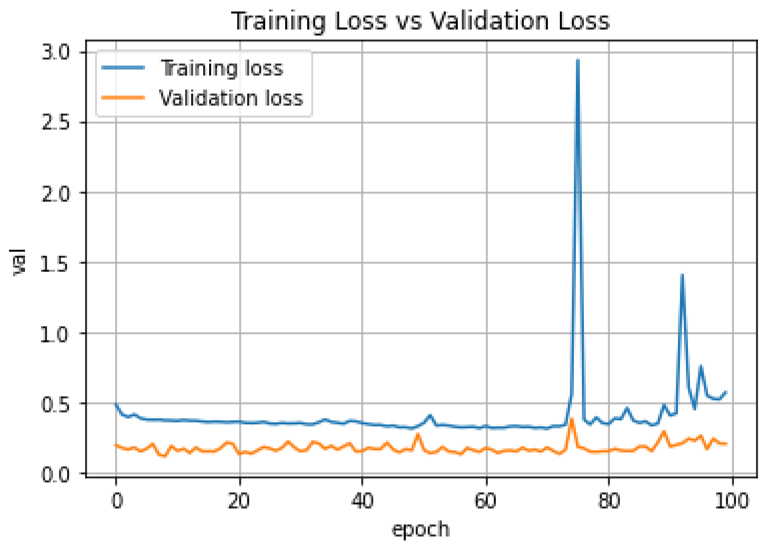
In [ ]:

```
layers = [0,1,2,4] #layer 0:lstm 1:lstm 3:dense

weights = {}
biases = {}
for layer in layers:
    weights[layer] = model.layers[layer].get_weights()[0]
    biases[layer] = model.layers[layer].get_weights()[1]

nlayer = 1
print(np.shape(weights[nlayer]))
print(weights[nlayer])

print(np.shape(biases[nlayer]))
print(biases[nlayer])
```

```
(64, 256)
[[-0.33196753  0.15261273  0.1911132  ... -0.11612164 -0.05420372
   0.2248396 ]
 [-0.06651883  0.096737   -0.00911993 ...  0.1204385  -0.03066622
  -0.11666579]
 [-0.01369409 -0.07260814  0.1339566  ...  0.06387173  0.00710219
  -0.0227774 ]
 ...
 [-0.03166063  0.15804914 -0.05268697 ...  0.08493868  0.10186508
  -0.02508203]
 [-0.05164647  0.06643188 -0.02214655 ...  0.11415598  0.00885071
   0.02114384]
 [-0.16861597 -0.2258982   0.01559666 ...  0.10478915 -0.0646678
   0.18086322]]
(64, 256)
[[-0.42380494  0.20582585 -0.46434513 ... -0.04303503 -0.26551652
  -0.15552069]
 [-0.20493926  0.30195326  0.09652823 ... -0.01944047  0.15859011
   0.08209214]
 [-0.37913728 -0.53568465 -0.09281334 ... -0.10309111 -0.5107119
   0.05467125]
 ...
 [ 0.09288114  0.12025551 -0.04001126 ...  0.02113777 -0.12093887
   0.08953512]
 [-0.01653129 -0.00240258  0.13873385 ...  0.00154706 -0.27371776
   0.22063132]
 [ 0.3088279  -0.15287729  0.1684417  ...  0.11152785 -0.36020002
  -0.10524259]]
```

In [ ]:

```
xx = np.arange(0,len(history.history['loss']))

"""print(history.history['loss'])
print(history.history['val_loss'])
print(xx)"""

plt.figure(1)
plt.plot(xx,history.history['loss'], label='Training loss')
plt.plot(xx,history.history['val_loss'], label='Validation loss')
plt.title("Training Loss vs Validation Loss")
plt.xlabel("epoch")
plt.ylabel("val")
plt.legend()
plt.grid()
```



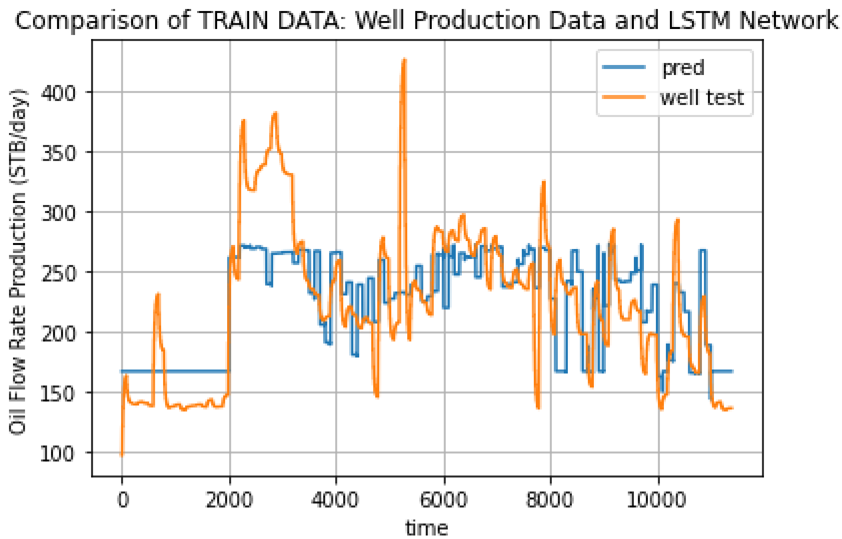
Predicting Values

In [ ]:

```
#Make prediction
#model = keras.models.load_model("RNN_Model")

n_days_for_prediction = 20
prediction = model.predict(trainX[:]) #shape = (n, 1) where n is the n_days_for_prediction

#Perform inverse transformation to rescale back to original range

prediction_copies = np.repeat(prediction, df_for_training_y.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)

print('nilai pred:',y_pred_future)

yy = scaler.inverse_transform(trainY)

x_axis = np.arange(0,y_pred_future.shape[0])

print(f"ukuran y: {np.shape(yy)} ukuran y pred: {np.shape(y_pred_future)}")

plt.figure(2)
plt.plot(x_axis,y_pred_future, label='pred')
plt.plot(x_axis,yy, label='well test')
plt.title("Comparison of TRAIN DATA: Well Production Data and LSTM Network")
```

```
plt.xlabel("time")
plt.ylabel("Oil Flow Rate Production (STB/day)")
plt.legend()
plt.grid()
plt.show()
```

```
356/356 [==============================] - 4s 9ms/step
nilai pred: [[166.51282]
 [166.51282]
 [166.51282]
 ...
 [166.51282]
 [166.51282]
 [166.51282]]
ukuran y: (11386, 1) ukuran y pred: (11386, 1)
```



Metric

In [ ]:
```
import math
from sklearn.metrics import r2_score

MSE = np.square(np.subtract(yy,y_pred_future)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)
```

```
r2 = r2_score(yy,y_pred_future)
print("\nR2 Value:")
print(r2)
```

```
Root Mean Square Error:
40.267503592689

R2 Value:
0.5859916566545849
```

Forecasting Value/Test Data

In [ ]:
```
#df2 = pd.read_csv("upsample.csv")
#df2 = df2.iloc[:,0:152]
x2 = df['glir11'].to_numpy()[int(split*len(data)):]
y2 = df['qo11'].to_numpy()[int(split*len(data)):]

#x2 = df2['glir11'].to_numpy()
#y2 = df2['qo11'].to_numpy()

#New dataframe with only testing data
x2 = x2.reshape(len(x2),1)
y2 = y2.reshape(len(y2),1)
df_for_testing_x = x2
df_for_testing_y = y2

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_testing_x)
scaler2 = scaler.fit(df_for_testing_y)

df_for_testing_scaled_x = scaler.transform(df_for_testing_x)
df_for_testing_scaled_y = scaler2.transform(df_for_testing_y)

#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 5. We will make timesteps = 14 (past days data used for testing).

#Empty lists to be populated using formatted testing data
testX = []
testY = []
```

```python
n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_testing_scaled has a shape (12823, 5)
#12823 refers to the number of data points and 5 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_testing_scaled_x) - n_future +1):
    testX.append(df_for_testing_scaled_x[i - n_past:i, 0:df_for_testing_x.shape[1]])

for i in range(n_past, len(df_for_testing_scaled_y) - n_future +1):
    testY.append(df_for_testing_scaled_y[i + n_future - 1:i + n_future, 0])

testX, testY = np.array(testX), np.array(testY)

print('testX shape == {}.'.format(testX.shape))
print('testY shape == {}.'.format(testY.shape))
```

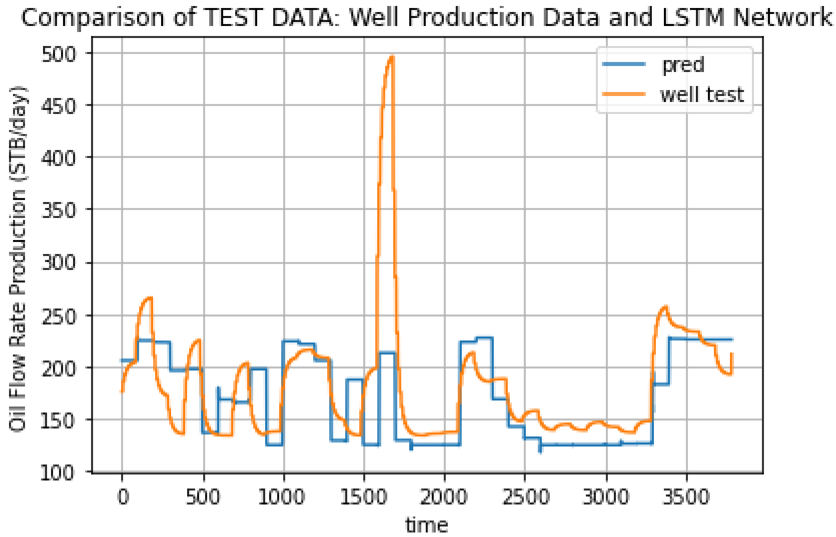
```
testX shape == (3787, 14, 1).
testY shape == (3787, 1).
```

In [ ]:

```python
#Make forecast
#model = keras.models.load_model("RNN_Model")

n_days_for_forecast = 20
forecast = model.predict(testX[:]) #shape = (n, 1) where n is the n_days_for_forecast

#Perform inverse transformation to rescale back to original range

forecast_copies = np.repeat(forecast, df_for_testing_y.shape[1], axis=-1)
y_fore_future = scaler.inverse_transform(forecast_copies)

#print('nilai pred:',y_fore_future)

yyy = scaler.inverse_transform(testY)

x_axis = np.arange(0,y_fore_future.shape[0])

print(f"ukuran y: {np.shape(yyy)} ukuran y pred: {np.shape(y_fore_future)}")

plt.figure(2)
plt.plot(x_axis,y_fore_future, label='pred')
plt.plot(x_axis,yyy, label='well test')
plt.title("Comparison of TEST DATA: Well Production Data and LSTM Network")
```

```
plt.xlabel("time")
plt.ylabel("Oil Flow Rate Production (STB/day)")
plt.legend()
plt.grid()
plt.show()
```

```
119/119 [==============================] - 1s 6ms/step
ukuran y: (3787, 1) ukuran y pred: (3787, 1)
```



Metric

In [ ]:
```
import math
from sklearn.metrics import r2_score

MSE = np.square(np.subtract(yyy,y_fore_future)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)

r2 = r2_score(yyy,y_fore_future)
print("\nR2 Value:")
print(r2)
```

```
Root Mean Square Error:
50.91016010881377

R2 Value:
0.2482057508163117
```